(51) Int. Cl.: G 06 f, 9/18

BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(52) Deutsche Kl.: 42 m3, 9/18

# (10) (11) Offenlegungsschrift 2150 506

(21) Aktenzeichen: P 21 50 506.6

(22) Anmeldetag: 9. Oktober 1971

(43) Offenlegungstag: 17. Mai 1973

Ausstellungspriorität: —

(30) Unionsprio ität

(32) Datum: —

(33) Land: —

(31) Aktenzeichen: —

---

(54) Bezeichnung: Verfahren zur Herstellung von Querverbindungen zwischen Programmen von Datenverarbeitungsanlagen während der Programmausführung

(61) Zusatz zu: —

(62) Ausscheidung aus: —

(71) Anmelder: IBM Deutschland GmbH, 7000 Stuttgart

Vertreter gem. § 16 PatG: —

(72) Als Erfinder benannt: Ploechl, Wilhelm, 7031 Rohrau

---

DT 2150506

© 5.73 309 820/325 22/70

ABSTRACT
1394258 Data processing systems INTER- NATIONAL BUSINESS MACHINES CORP 6 Oct 1972 [9 Oot 1971] 46152/72 Heading G4A A data processing system including an inter- nal store or stores which store program modules to be executed under the control of a supervisor program is arranged to provide automatic cross-referencing between the modules. Each module has a first table EXTAB comprising external symbols EXTRN defined in at least one other module and address fields corresponding to each of these symbols, which fields are initially blank, and a second table ENTAB comprising internal symbols ENTRY defined by the module and the addresses of these sym- bols. The supervisor program has a reference table ADTAB defining the addresses of the first and second tables. When a module invokes an external symbol within its first table and the corresponding address field is blank, it calls the supervisor program which searches the second table of the modules, using addresses from the reference table, to obtain a symbol address which is then entered into the corresponding address field in EXTAB. When a module is to be removed from the system the cross-references to it are erased.

CLAIMS

**WARNING** start of CLMS field may overlap end of DESC **.

LEAC instruction is available, this address is then checked in a number of check steps.

Step 132 checks whether the address of the second operand is a valid address, i.e., whether it is not outside the permissible range of addresses of the data processing system.

If it is not, an address field of type I (invalid) is indicated by step 133. Alternatively, i.e., if the address is a valid address, step 134 determines whether the second operand begins on a full word boundary. Step 135 provides an addressing error indication if step 134 yields a " no " indication. This type of addressing error is classified as a type II error. A "yes" indication leads to a further check step 136, which determines whether the address found is a protected address. If so, step 137 provides a type III addressing error indication, which, in turn, leads to a storage protection interrupt of the current program. Alternatively, step 138 is next im implemented, this checks the operation code of the instruction (bit positions zero to seven) to determine whether it contains "LEAC".

If it does not, step 139 checks the presence of other OP codes by means of operations not shown in detail. These operations are of no interest in this connection, since it is assumed that the instruction to be interpreted is a LEAC instruction. A "yes" indication in step 138 results in the termination of the interpretation cycle and entry into the execution (E) cycle. The first step of this cycle step 140 consists of reading in the second operand. In step 141 the second operand is checked for zero. If the value found is other than zero, the second operand, in step 142, is transferred to register R1 which represents the first operand of the LEAC operation.

Thus, the operation "Load Address" is executed, and the machine can begin to interpret the next instruction. If on the other hand, step 141 yields a " yes", step 143 leads to an interrupt code 0020 to be transferred to a corresponding register (not shown), which, by step 144, causes the current program to be interrupted. The program interrupt of step 144 is identical to the program interrupt of step 155 in Figure 7A and leads to the supervisor program being invoked to execute the operations as previously described with reference to Figure 7B.

The operations as explained by means of
Figure 9 can be executed utilising a suitable microprogram on known microprogrammed IBM System/360 machines. A system of this type is described, for example, in British Patent Specification No. 1,242,437.

In the embodiment as described above, the tables EXTAB and ENTAB were formed in a single store containing the respective problem programs. This store would normally be main storage 18. It may, however, be advantageous in the interest of high operat
ing speeds to store the tables in one or several stores operating at particularly high speeds and which respond to instructions referring to ENTRY or EXTRN symbols. The same holds for address table ADTAB, with the corresponding working storage being controlled by the supervisor program. Data processing systems which, in addition to main storage, comprise high-speed working stores or universally usable register sets are known from the art, as may be seen, for example, from the above mentioned British Patent
Specification.

The arrangements are also suitable for computers which do not use an instruction for externally defined addresses similar to the
LEAC instruction. In such cases the following instruction sequence is to be inserted into the code listings (e.g. listing 1) instead of the
LEAC instruction:

```
L 10, =A(EX)
LTR 10, 10
BNZ *+6
SVC nnn
```

These instructions forming part of the standard instructions of IBM System/360 data processing equipment serve the following purpose. By means of the L instruction an address specified more specifically in the second operand is loaded into a register specified by the first operand.The second instruction is a "Load and Test" instruction, by means of which the contents of register 10 are reloaded into this register, with various test operations referring to the numerical1 value to be loaded being executed. One f these test operations checks whether the value to be loaded is zero. Instruction BNZ refers to a branch that is taken if the tested or checked value is other than zero. This leads to the next instruction being skipped if the value tested by means of the LTR instruction is other than zero. Alternatively, instruction SVC is implemented which causes a branch to be taken to the supervisor program. The operations executed by these in structions have the same effect as those effected in response to the LEAC instruction.

WHAT WE CLAIM IS:-

1. A stored program data processing system including an internal store or stores arranged to store program modules which are to be executed under the control of a supervisor program, in which each program module has associated therewith a first table comprising external symbols defined in at least one other module and address fields corresponding to each of these symbols, which fields are initially blank, and a second table comprising internal symbols defined by the module and the addresses of these symbols, and the supervisor program has associated therewith a reference table defining the

addresses of the first and second tables, and in which system, in operation, when a program module invokes an external symbol within its first table and the corresponding address field is blank, it calls the supervisor program which, in response thereto, searches the second tables of the modules, using addresses from the reference table, to obtain the symbol address which is then entered into said corresponding address field.

2. A svstem as claimed in claim 1 in which each entry in the second tables includes a marker field into which a use mark is entered when the entry is first used as a cross-reference between modules, the use marks serving to indicate use of the entry when existing cross-references are to be erased upon removal of a program module from the system.

3. A system as claimed in claim 2 in which, upon removal of a program module from the system, the second table associated therewith is searched for each entry including a use mark, and, by means of the symbols stored therein the first tables of the other modules are searched for these symbols and any found are erased.

4. A system as claimed in any of the previous claims in which the reference table comprises a program module identification entry associated with each pair of first and second table addresses.

5. A system as claimed in any of the previous claims in which the reference table includes, for each pair of addresses of first and second tables associated with a program module, a marker field into which indicator data is entered when one external address has been entered into the corresponding first table or when the corresponding second table has been referenced at least once.

6. A stored program data processing system when programmed to operate in a manner substantially as described herein with reference to Figures 2 to 10 of the accompanying drawings.

DESCRIPTION

(54) STORED PROGRAM DATA PROCESSING SYSTEM
(71) We, INTERNATIONAL BUSI
NESS MACHINES CORPORATION, a
Corporation organized and existing under the laws of the State of New York in the United States of America, of Armonk, New York 10504, United States of America, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:~
This invention relates to stored program data processing systems, and in particular to such systems when employing a plurality of cross-referenced program modules.

A program used to solve a data processing problem by means of an electronic computer (problem program) consists in most cases of several autonomous program components which may be prepared independently of each other. Each program component which is also referred to as a program module comprises a set of instructions for cross-referencing parts of other program modules. Such module parts may take the form of subroutines, constants, intermediate results, etc., which are defined in the respective module where they are used or generated. Referencing may be effected via actual storage addresses at which the corresponding parts of the other program modules are stored. In order to do this, however, the actual storage addresses must be known when the program components are prepared.Any cross-referencing between program components is thus finally determined by the programmer, but this is only possible where the number of program components to which storage space has to be assigned can he planned in advance.

Cross-referencing of instructions of program modules may also be effected by means of symbols representing the names of program components to be referred to. This facilitates programming in so far as the programmer does not have to work with the actual storage addresses which the program

components will occupy later. By means of a linkage (editor) program the symbols are associated with actual storage addresses and are inserted into the individual program components. The linkage program generates a loadable problem program which is stored in an external storage of the data processing system until it can be loaded into internal storage for execution (IBM Systems Reference Library GC 2865342 " IBM System/360 Operating System - Introduction " -- October 1969, pages 77 to 79).This leads to programming being facilitated and a high degree of flexibility being obtained. Before the loadable problem program is made available, however, a separate run of the linkage program is required, by means of which the program components executed concurrently in internal storage are referred to each other. In addition, the linkage program has to be rerun every time the program component structure within the problem program is changed. This makes it difficult to prepare problem programs in which the program component structure is changed as a function of the respective requirements (dynamic program structure), rather than according to a given plan. During the execution of the linkage program the data processing system is not available for running problem programs, which leads to the user time being reduced.

According to the invention, there is provided a stored program data processing system including an internal store or stores arranged to store program modules which are to be executed under the control of a supervisor program, in which each program module has associated therewith a first table comprising external symbols defined in at least one other module and address fields corresponding to each of these symbols, which fields are initially blank, and a second table comprising internal symbols defined by the module and the addresses of these symbols, and the supervisor program has associated therewith a reference table defining the addresses of the first and second tables, and in which system, in operation, when a program module invokes an external symbol within its first table and the corresponding address field is blank, it calls the supervisor program which, in response thereto, searches the second tables of the modules, using addresses from the reference table, to obtain the symbol address which is then entered into said corresponding address field.

In order that the invention can be fully understood, a preferred embodiment thereof will now be described with reference to the accompanying drawings, in which:
FIGURE 1 is a block diagram illustrating the known sequence in which different programs are cross-reference by means of a separate linkage program;
FIGURE 2 is a modification of the block diagram of Figure 1 in which the linkage program is removed;
FIGURE 3 is a simplified flow chart showing program cross-referencing;
FIGURES 4A, B are structures of the reference and address tables used in the flow chart of Figure 3;
FIGURES SA, B, C are representations of the status of the reference and address tables of Figures 4A, B when, for example, a program P1 and a program P2 are crossreferenced;;
FIGURE 6 is a flow chart for setting up the address table ADTAB in the supervisor program during the loading of the programs into the internal storage of the data processing system;
FIGURES 7A, B are detailed flow charts for cross-referencing two programs P1, P2;
FIGURES 8A, B are flow charts for erasing existing cross-references when one of the progorams is removed from the internal storage of the data processing system;
FIGURE 9 is a flow chart of the micro program steps pertaining to the LEACH instruction; and
FIGURE 10 is a representation of the format of the LEAC instruction.

The flow chart of Figure 1 shows the orthodox way in which three program components are cross-referenced to form a loadable problem program. Each program component is written in a suitable programming language and is then punched into a card deck 12. The program punched in the card deck is referred to as a source module QM. Each source module QM 1 to QM3 is translated into an object module OM in a separate translation step 13. The object module is an executable program component containing the operation codes and the addresses in the language of the computer. The translation step 13 is carried out by means of suitable translation programs TR-PRGR 1 to 3. The object modules OM1

to OA#3 are temporarily stored in an external storage device 14 of the computer system.In a further job step 15, under the control of a linkage program, object modules OM1 to OM3 are combined to form a loadable object program LOP which is temporarily stored in a further external storage 16. Relative addresses assigned to the corresponding parts of other object modules are inserted by the linkage program into the instructions of those object modules referring to such parts. From temporary storage 16, serving as the computer's library storage for loadable object programs, the corresponding object program LOP is loaded in a load step 17 under the control of a load program into main storage 18 of the computer. The program LOP in main storage 18 to which the actual main storage addresses were assigned by the load program can sub sequently be executed by the computer.

The flow chart of Figure 2 deviates from that of Figure 1 in so far as step 15 and external storage 16 are omitted. In step 13, source modules QM1 to QM3 punched in card decks 12 are translated by means of translation programs TR-PRGR 1 to 3 into object modules OM1 to 3 which are stored as load modules LM1 to 3 in external storage device 14. Storage device 14 serves as a library for loadable and executable program components. In load step 17 the load program causes load modules LM1 to 3 to be successively loaded into main storage 18 of the computer. During this process each load module LEM 1 to 3 is assigned main storage locations of its own. Program components LM1 to 3 in main storage are not yet crossreferenced. The cross-references to be made are merely symbolically marked.In each program component the symbols defined by another program component are marked "EXTRN", whereas the defined symbols of each program component to which reference is made by other program components are designated as "ENTRY". At the beginning of each load module LM1 to 3 the translation programs of step 13 set up a first table of all EXTRN symbols and a second table of all ENTRY symbols in accordance with the information of source modules QM. In addition, as load modules LM1 to 3 are loaded into main storage an address table is set up in the supervisor program for the addresses of all ENTRY and EXTRN tables. Subsequently, the computer can begin executing the program, with program components LM1 to 3 in main storage 18 jointly forming the object or problem program.

Figure 3 shows the steps for cross-referencing programs in main storage of the computer. During program translation a first table EXTAB and a second table ENTAB are set up in step 21 at the beginning of each load module. The structure of these tables is shown in Figure 4A. Table EXTAB comprises a header line with the entries A and
GEX. Entry A requires one byte and is indicative of a table EXTAB. If A = 1, then the header line is followed by entries of the
EXTAB table, whereas when A=0, there are no further table entries. Entry GEX requires 3 bytes and indicates the number of elements (lines) contained in table EXTAB. The subsequent lines of this table each consist of a field NAMEX and a field ADREX.The former field is eight bytes and contains the name of a svmbol included in the corresponding program component and which is defined in another program component. Field ADREX is three bytes which are initially zero and which later accommodate the address of the externally defined symbol. Table EXTAB comprises as many lines as are specified by the value in field GEX. Table ENTAB includes, in a header line field, GEN which has three bytes and which defines the number of elements contained in this table. The following lines of table ENTAB each comprise three fields: NAMEN, B, ADREN. Field
NAMEN consists of eight bytes and contains the name of a symbol defined in the corresponding program and used in another program. Such symbols are hereafter generally marked by an entry symbol.Field B consists of one byte which is initially zero and which is set to one when the associated field
ADREN is used. Field ADREN requires three bytes and contains the address of the symbol entered in the associated field NAMEN. The hatched fields of the two tables contain no information.

In step 21 of Figure 3 tables EXTAB and ENTAB are thus set up for each program. The entries for fields NAMEX of table

EXTAB and fields NAMEN and ADREN of table ENTAB are derived from the information in the corresponding program. In this program all externally defined symbols are designated as EXTRN, an entry being made in table EXTAB for each symbol thus designated, with fields ADREX of this table being left blank. Similarly, the symbols invokable from other programs are designated as ENTRY. For each symbol thus designated an entry is made into one of the fields NAMEN of table ENTAB during program translation.

In addition, the address of the corresponding ENTRY symbol, which is specified in the program, is stored in this field, which fields B remaining in the zero state.

In the subsequent step 22 which is implemented during load operation 17 of Figure 2 an address table ADTAB for tables EXTAB and ENTAB included in the program components pertaining to the problem program is set up by the supervisor program under whose control the problem program is executed. The structure of table ADTAB is shown in Figure 4B. Each line of this table comprises five fields as follows: IDENT, E, AEX, F, AEN. Field IDENT has eight bytes and contains the name of a program which has been loaded into main storage 18 of the computer and which includes tables EXTAB and/or ENTAB. Field E has one byte which is initially zero. This field is set to one when table EXTAB contains at least one EXTRN symbol in the program specified in field IDENT.Field AEX has four bytes and contains the address of table EXTAB which belongs to the program whose name is specified in the associated field IDENT. Field AEX is zero when no table EXTAB exists for the program. Field F has one byte which is initially zero. Into this field the value one is stored when table ENTAB pertaining to the program specified in field IDENT has been used at least once during the execution of the problem program. Field AEN consists of four bytes and contains the address of table ENTAB which belongs to the program whose name is in field IDENT. Field AEN is zero when the corresponding program does not include a table ENTAB.

Table ADTAB is set up by the supervisor program which controls the execution of the problem program in the computer. The supervisor program is also operative to crossreference program components LM1 to 3 forming the problem program and stored in internal storage 18. If during the execution of the problem program (step 23 in Figure 3) an instruction occurs invoking an EXTRN symbol, it is attempted to load the address of this symbol from table EXTAB of the program into the corresponding instruction.

To this end it is checked in step 24 by means of the " Load Address " instruction LEAC R1, D2 (X3, B3 ) whether there is an address entry for the corresponding EXTRN symbol in table EXTAB. In this instruction LEAC represents the symbolic operation code, R, a register address as the first operand of the instruction, and D2(X2, B2) the second operand which is an address made up of the contents of an index register X2, the contents of a base register B2, and a displacement address D2. The operation implemented as a result of the instruction consists in the second operand being loaded into the register indicated by the first operand.Such an instruction could be included, for example, in existing data processing systems of IBM System/360, Model 25, by storing a microprogram whose steps will be described in detail in connection with Figure 9.

During execution of the LEAC instruction it is checked whether the second operand specified in the program as the address of the corresponding EXTRN symbol has a value of zero. If it turns out in step 24 of Figure 3 that the required address has a value other than zero, this indicates that the address concerned is in table EXTAB. Therefore, after transmission of the address into the register defined by the first operand the program is continued in accordance with step 23. On the other hand, if it is found in

step 24 that the required address has a content of zero, the program is interrupted in step 25, and the supervisor program of the computer is invoked. The supervisor program, using the corresponding EXTRN symbol as a search argument, then searches the existing ENTAB tables for a name field of the same contents (step 26).To this end the supervisor program uses table ADTAB in order to find, for each program component LM, the ENTAB table whose address is stored in the right-most field AEN of table ADTAB. If during the search of the ENTAB tables a field NAMEN is found whose contents are equivalent to those of the EXTRN symbol, the contents of the associated field ADREN containing the address of the ENTRY symbol having the same name are transferred to table EXTAB of the interrupted program. In step 27 (Figure 3) the EXTRN symbol which caused the interrupt is transferred to field ADREX. After this step has been completed the program continues in accordance with step 23 from the point at which the interrupt occurred. In accordance with step 24, it is again tried by means of the LEAC instruction to find the address of the external symbol in table
EXTAB. As the required address is in this table, the fresh attempt is successful and the program is continued, using parts of another program module LM, which were defined by the EXTRN symbol. Upon occurrence of the next EXTRN symbol steps 24 to 27 are reexecuted as described.

Figures 5A to C and the following code listings illustrate a numerical example of the steps of Figure 3. In this example a program P1 invokes three external symbols EXTRN
R, S, T which are defined in a program
P2. The code listings shown are in IBM
System/360 assembler language.
Program P1 (listing 1)

```
Name OP Code Operand Comments
P 1 START
only EXTAB available::EX=l(bit 0)
GEX DC AL3(3) GEX = 3
DC CL8'R' NAMEX (1)
AR DC FlO? EXTAB ADREX(l)
DC CL8'St NAMEX (2)
AS ¦DC F'0' ¦ ADREX (2)
DC DC CL8"T" NAMEX (3)
AT DC F'0' J ADREX (3)
EXTRN R S, T
1
LEAC 10,AmAR' Load register 10 with address of R.


lO,AR
L 3, 0 (10) Use register 10.


LEAC 8, AS
LEAC 1,AT
END
```

In accordance with this listing, which shows program P1 as a source module, the translation program (step 13 of Figure 2) sets up an element of table EXTAB in program P1 for each external symbol R, S, T. This table which is shown in Figure 5A is stored at address 10 000 in main storage 18 of the

computer and consists of three elements to which the symbols R, S and T are assigned in the NAMEX fields (Figure 4A). Before execution of the program the ADREX fields of these elements have a content of zero. The fields marked b (Figure 5A) contain no significant information. In the header line of table EXTAB entry A is set to "1" to indicate the existence of a table EXTAB.This is effected by means of line 2 of the listing by the declaration of a hexadecimal constant 80 which sets the highest order bit position of the corresponding field to the value one.

The next statement sets field GEX in the header line of table EXTAB to the value three; since the table contains three elements.

The next six statements declare elements R,
S, T for table EXTAB. For simplicity, it is assumed that program P1 does not include an ENTAB table.

Program P2 (listing 2)

```
Name OP Code Operand Comments
P2 START
A DC X'40' only ENTAB available:EN=l(bit 1)
GEX DC AL3(0) no EXTAB elements: GEX=0
DC X'OO' i field not being used: 0
GEN DC AL3(3) GEN=3
fDC CL8'R' NAMEN (1)
DC Xlool 3(1)
DC AL3(R) ADREN (1)
DC CL8'S' ENTAB NAMEN (2)
X'OO' B (2)
DC AL3(S) ADREN (2)
DC CL8'T' NAMEN (3)
DC X'OO' 5B (3)
DC AL3 ( T) ADREN (3)
ENTRY R, S, T
R DS F
S DC CL100
T DS D
END
```

The initial statements of this listing define table ENTAB in program P2 (Figure SA).

The header line of this table is at main storage address 16 000. Upon the insertion of a hexadecimal constant 40 into field A, bit position 1 of this field is set to a value of one (statement 2 of the listing), so introducing mark Yen=1. This means that program
P2 comprises only a table ENTAB and no table EXTAB. Accordingly, field GEX of the header line is set to zero by means of the third statement. In the second line of table ENTAB (header line of this table) a zero is entered into field A, as this field will not be used subsequently. The fifth statement of the listing causes the value three to be declared for field GEN of table ENTAB.

In the succeeding nine statements the three elements R, S, T are declared. During translation in accordance with step 13 of Figure 2 this results in a table structure as shown in Figure 5A. As the translated program P2 is loaded in accordance with step 17 of

Figure 2, main storage addresses 17 000, 17 004, 17 104 are inserted into address fields ADREN of the elements of table ENTAB.

During the same operation, i.e. during loading, table ADTAB in accordance with Figure 5A is set up by the supervisor program. In the course of this, main storage address 10 000 is entered into address field AEX (address of table EXTAB) associated with the program name P1. Similarly, main storage address 16 004 is entered into address field
AEN (address of table ENTAB) associated with the program name P2. In line 10 of listing 1 (Program P1) the symbols R, S, T are specified as external symbols. The succeeding lines 11 to 13 refer to parts of program P1 which are not shown. Line 14 contains the LEAC instruction, by means of which the address of the external symbol R is to be loaded from table EXTAB into register 10.

During the execution of the program this instruction causes an interrupt, since the address field of element R in table EXTAB is blank when the LEAC instruction is encountered for the first time. The changes resulting in the table during this program interrupt are shown in Figure 5B. Via the second element of table ADTAB the supervisor program locates address 16 004 of the header line of table ENTAB in program P2. Subsequently, the supervisor program begins to search fields
NAMEN of this table for symbol R, finding it in the first element of the table. Then main storage address 17000 in field ADREN is transferred to field ADREX of the first element of table EXTAB in program P1. This address field is associated with the external symbol R specified in the LEAC instruction.

In addition, a one is entered into field B of the first element in table ENTAB to indicate that this element was previously used in the program. In field E of the first line of
ADTAB a one is stored to indicate that at least one address is stored in EXTAB of P1.

A one is also stored in field F of the second line of ADTAB to indicate that ENTAB has been used by P2 at least once. Program P1 is continued by a fresh, and this time successful, attempt to load the address of symbol R from table EXTAB into register 10. In the following line 15 this address fetched from program P2 is used in program P1. Program P1 and program P2 are thus cross-referenced under the control of the supervisor program during the execution of program P1. Corresponding cross-references are effected by the
LEAC instructions in lines 19 and 24 of listing 1. The resultant changes in tables EXTAB of program P1 and ENTAB of program P2 can be seen from Figure 5B. The remaining lines of listing 1 concern parts of program P1 which have no bearing on the present invention. The same holds for lines 16 to 22 and 27 to 29 of listing 2.In lines 24
co 26 of listing 2 symbols R, S, T are defined which are externally mvokable in program PZ.

The above example merely refers to unidirectional cross-referencing from program
P1 to program P2. In accordance with the same principle, it is possible to cross-reference in the reverse direction, i.e. from program P2 to program P1. In this case programs P2 and P1 would include suitable definitions for setting up m EXTAB and an
ENTAB table, respectively. Reciprocal crossreferencing during the execution of the programs is not limited to 2 programs; this kint of cross-referencing can be extended to a larger number of programs which are concur rently stored in main storage 18 of the computer. To achieve this, address table ADTAB (Figure 5A) used to address tables EXTAB and ENTAB in the various programs can thus be enlarged as necessary.

The steps implemented by the supervisor program of the computer when the program s loaded into main storage in order to crossreference another program concurrently in main storage and also when one of the programs is to be removed from main storage will now be explained with reference to

Figures 6 to 8. The operations to be executed by the supervisor program when a program is loaded are shown in Figure 6. Block 31 generally refers to the loading of the program at address ADR. In step 32 a check is made to determine whether tables EXTAB and
ENTAB in field A (Figure 4A) at the beginning of the program contain indicator values of EX=0 and Yen=0. If both values are zero the program, in accordance with step 30, returns to the load program which subsequently transfers the individual instructions of the corresponding problem program to the assigned main storage locations. If on the other hand, one of the four EX and EN reserved bit positions contains a value other than zero, step 33 determines whether there are any blanks left in address table ADTAB (Figure 4B). The number of occupied elements in this table is denoted by the symbol N, and MAX represents the permissible number of elements in table ADTAB. If the condition
N < MAX is not fulfilled, the program, by means of step 34, returns to the load program.

Information indicating that for space reasons externally defined symbols could not be crossreferenced is then stored in the program that caused the load program to be implemented.

Alternatively, the control of the supervisor program proceeds with steps 35 which sets a run value of i=l. The run value i denotes the element in table ADTAB which is being processed. Step 36 checks whether element in table ADTAB, i.e. in the present case the first element of this table, contains a blank field IDENT. If the first element is already occupied, the answer will be "no", so that step 37 causes run value i to be incremented by one. Step 38 checks whether the table size has been exceeded. If so, a system error has occurred, and this is indicated at step 39. If, on the other hand, value i does not exceed the table size, the program branches back to step 36. If step 36 indicates that the checked IDENT field is blank, step 40 causes the value N, denoting the occupied elements of table ADTAB, to be incremented by 1.

Subsequently, in step 41, the name of the program to be loaded is transferred to field
IDENT. The supervisor program checks in step 42 whether the EX indicator in table
EXTAB of the program to be loaded is one.

If so, step 43 causes the address of the header line of table EXTAB to be transferred to address field AEX of the element of table
ADTAB which is currently processed. As this table is normally at the start of the program to be loaded, the address concerned is the starting address ADR of this program. Control then proceeds to step 44 which checks whether the indicator value EN of table
ENTAB of the program to be loaded is one.

If EN has a value other than one, a system error has occurred, this is indicated by step 39. Alternatively, step 45 leads to the address field of the element currently processed to be loaded into table ADTAB. The address to be loaded is derived from
ADR+4+ 12 * GEX.

This expression means that the starting address ADR of the program to be loaded is incremented by the storage space occupied by table EXTAB. Thus, table ENTAB immediately adjoins table EXTAB. After step 45, step 46 returns to the load program which subsequently continues the load operation.

Cross-referencing of programs that are concurrently in main storage of the computer is shown in Figures 7A and 7B. Step 51 of
Figure 7A concerns the invokation of an externally defined symbol in a program executed in the computer. As can be seen from listing 1, this invokation is performed by means of the LEAC instruction which loads the address of its second operand into a register designated by the first operand. During this operation various check steps are implemented of which only steps 52 and 53 are shown in Figure 7A.

Step 52 determines whether the value of the address of the second operand to be loaded is zero. If this value is found to be other than zero, step 53 checks whether the address concerned is a protected address. If not, the program is resumed after further check steps have been implemented. Alternatively, the program is branched to step 54. A "yes" indication in step 52 means that the address of the required externally defined symbol is not contained in the EXTAB table of the program. The program interrupt in accordance with step 54 causes branching to the supervisor program, and this, at step 56, initiates the cross-reference routine shown in Figure 7B. In step 57 the address Q of operand
OP2 of the LEAC instruction is fetched.This address referring to an ADREX field of table
EXTAB of the interrupted program is decremented by eight in step 58, so locating the address of field NAMEX pertaining to this field. The contents of this field, which correspond to the invoked symbol, are subsequently used as a search value. In step 59 a run value of i=l is set. This values denotes the number of the element in address table
ADTAB which has just been invoked. In step 60 it is checked whether field IDENT of element i in table ADTAB is blank. If so, run value i is incremented by one in step 61. Step 62 checks whether run value i exceeds the size of table ADTAB. If it does not, a branch is taken to step 60.If, after steps 60 to 62 have been repeated several times, it is determined by the supervisor program that the table size has been exceeded, control is passed to the problem program via step 63 which indicates that there is no symbol of the required name in main storage of the computer. If the answer yielded by step 60 is "no" meaning that element i in table ADTAB is occupied, step 64 becomes effective. This step checks whether address field AEN pertaining to the same table element has a content other than zero, and at the same time determines whether the program associated with this element via the IDENT field comprises a table ENTAB. If step 64 yields a "yes", step 65 causes a run value J = 1 to be set which denotes the element being processed in the respective table
ENTAB Alternatively, a branch is taken from step 64 to step 61.The supervisor program then checks in a search loop, comprising steps 66, 67, and 68, whether the name of the required symbol is included in table
ENTAB. Step 66 checks whether the name of element J equals that of the search value defined in step 58. A no result of this comparison causes run value J to be incremented by one in step 67. Step 68 determines whether the run value exceeds the size of table ENTAB. If it does not, step 66 is repeated; alternatively, a branch to step 61 is taken, leading to table ENTAB of another program being checked.

If it is indicated in step 66 that the contents of the NAMEN field are identical to the search value, step 69 is implemented.

This causes the contents of field ADREN in element J of the searched ENTAB table to be transferred to field ADREX which is at address Q and is associated as an address field with the search value (EXTRN) symbol).

In step 69, field E of the invoked element i in table ADTAB is set to one to indicate that at least one address has been stored in the EXTAB table associated with this element.

Then step 70 is implemented, by means of which element B in the searched table ENTAB is set to one to indicate that address field
ADREN of this element is being used. In addition, step 70 causes field F of element i in address table ADTAB to be set to one, thereby indicating that table ENTAB associated with this element has been used at least once. Step 71 then passes control back to the problem program. The steps as explained result in the address of the external symbol being transferred to table EXTAB. By repeating the LEAC instruction that previously caused the interrupt the problem program is continued.

If one of the programs contained in main storage 18 of the computer and which was cross-referenced in the course of the program is to be removed from main storage, the crossreferences between the programs

remaining in storage have to be dissolved to avoid addressing errors during the further execution of the programs. The steps required for this operation are shown in Figures 8A and 8B. Step 76 performs all those supervisor program operations which cause the program loaded at address ADR to be removed from main storage 18; these are not significant with respect to the present invention. For processing address table ADTAB run value i is set to one by means of step 77. In step 78 field
IDENT in element i of this table is invoked, and a check is made to determine whether its contents are identical to the name of the program to be removed.If not, run value i is incremented by one by step 79. This is followed by a check step 80 which determines whether i exceeds the table size. If it does not, step 78 is repeated and, if necessary, steps 79 and 80, too. If in the course of this operation, table ADTAB is fully processed, without a field IDENT with the name of the program to be removed being found, the cross-reference routine is terminated by step 114. In this case the program includes no EXTAB or ENTAB tables. An indication of concurrence in step 78 causes step 82 to be initiated, this causes the found IDENT field to be erased. Thereafter the value N, representing the number of elements occupied in table ADTAB, is decremented by one in step 83. Step 84 checks whether $N < 0$. If so, a system error is indicated in step 85.If not, field AEN in element i of table ADTAB is checked in step 86 to determine whether this field has a value other than zero. A no result from this check step leads to the reference dissolve routine being terminated by step 115, as in this case the program did not include an ENTAB table. A "yes" result of step 86 leads to step 87 which checks whether field F of element i has a contents other than zero. A "no" result leads to the cross-reference dissolve routine being terminated by step 116, since table
ENTAB associated with element i has not been used. A "yes " result of the check of step 87 leads to step 88 which sets run value J to one, with address AEN (i) being used as a base for the subsequent addressing of the elements in table ENTAB.Step 89 checks whether field B of element J in the invoked table ENTAB has a value other than zero. A "no" result of the check indicates that the ENTRY symbol associated with the corresponding table element has not been cross-referenced. In step 90 the J value is incremented by one, whereas step 91 checks whether the incremented J value exceeds the size of the processed ENTAB table. If it does not, step 89, and, if necessary, steps 90 and 91 are repeated. Alternatively, the whole ADTAB element i is erased, with the cross-reference dissolve routine being subsequently terminated by step 117.

If step 89 yields a "yes " result, which indicates that element J has been previously cross-referenced, step 94 (Figure 8B) is implemented. This leads to a new run value
L= 1 being set. This run value is used to process the IDENT fields of table ADTAB.

Step 95 checks whether field IDENT of element L in this table is blank. If so, value
L is incremented by one in step 96. Step 97 checks whether value L exceeds the size of table ADTAB. If it does not, a branch is taken to step 95. Alternatively the supervisor program branches to step 90 of Figure 8A, in which run value J for the processing of the invoked ENTAB table is incremented.

If it is found in step 95 that field IDENT of element L in table ADTAB is not blank, check step 102 is implemented. This step, by checking field AEX in element L for a value other than zero, determines whether an
EXTAB table exists for this element. A "yes" result leads to a further check step 103 which, by checking field E in the same element, determines whether an address has been previously stored in this EXTAB table.

A "no" result of steps 102 or 103 results in a branch to step 96 which causes run value L to be incremented. A "yes" result from step 103 leads to control proceeding to step 104 which defines a further run value K 1 and which sets an indicator switch
AZSCH to the check criterion zero. Run value K serves to process the element in the invoked EXTAB table, and the switch is used to interrogate address fields ADREX in this table. In the succeeding steps

the name of the ENTAB element whose B field has a contents other than zero (step 89 in Figure 8A) is used to check the existing EXTAB tables, with the name of the ENTAB element employed as a search argument. If a field of the same name is found in a field NAMEX of the EXTAB table, the associated address field is erased.These operations begin with step 105, by means of which the contents of field NAMEN in element J of table
ENTAB of the program to be removed from storage are compared with the contents of field NAMEX in element K of the EXTAB table being processed. If this comparison yields concurrence, address field ADREX associated with field NAMEX is erased in step 106.

The subsequent step 107 leads to value K being incremented by one, with step 108 checking whether the K value exceeds the slze of the EXTAB table being processed. If it does not, step 105 is repeated. If step ll)s yields a "no" result, step 109 checks whether address field ADREX of element K in table EXTAB contains the value zero. If it does, step 107 is implemented next. Alternatively, indicator switch AZSCH is set to one in step 110, this leads to an indication being stored that element K of table EXTAB, which was processed last, contains an entry.

Upon completion of step 110, step 107 is implemented. If the operation in accordance with step 108 yields a "yes" result, step 111 checks whether switch AZSCH is in the one state. If so, the checked table EXTAB contains at least one occupied address field
ADREX, whose associated field NAMEX d}cs not concur with the contents of argu mUnt field NA MEN in element J of table
ENTAB in the program to be removed from storage If step 111 yields a " no ", field E of element L in address table ADTAB is set to zero in step 112. Steps 111 and 112 are followed by a branch to step 96, to cause the program to proceed to the next element of the address table.

In summary, the above described steps cause all cross-references, which were made from other programs to the program to be removed from main storage of the computer, to be erased. The EXTAB tables of the other programs are searched for such crossreferences and the cross-references found are erased (step 106). In addition, the element of address table ADTAB associated with the program to be removed is determined and erased (steps 82 and 92). The result of these operations for the numerical example as described is shown in Figure 5C, and the object is to remove program P2 from main storage and to erase cross-references existing between program P2 and program 1)1 which remains in main storage.

The operation steps to execute the LEAC instruction in an IBM System/360 data processing system will be explained in detail by means of Figure 9. For carrying out these steps an IBM System/360, Model 25 using a microprogram may be employed, for example.

The execution of the LEAC instruction is preceded by the interpretation (I) cycle which does not differ from the interpretation cycle of other instructions of the same type.

By means of step 121 the bytes zero and one of the instruction to be interpreted are read from main storage 18 into an instruction register (not shown). Subsequently, the instruction address is incremented by two by step 122. Byte zero of the instruction contains the operation code. Step 123 checks whether the instruction of the operation code read is an instruction of the RX type. As the LEAC instruction belongs to this type, an interrogation for the presence of other instruction types, as shown by block 124, is not needed in the present case where the answer yielded is "no". If step 123 yields a "yes" result, bytes two and three of the instruction are read in step 125, and the instruction address is then again incremented by two in step 126.In step 127 the base register to be defined by the instruction to be interpreted is checked for the value zero.

The register concerned is the base register denoted by the B2 component in the LEAC instruction (Figure 10). This component is in bit positions 16 to 19 of the instruction, which are occupied by the first half of byte 2 (i.e. the third byte in the instruction).

If the contents of the base register are zero, step 128 checks whether the address of the second operand is equal to the displacement address in address field D2. If not, step 129 causes the address of the second operand to be made equal to the contents of base register B2 plus displacement address D2. In either case step 130 is executed next. This step checks whether the contents of register X2 are zero. Index register X2 is selected by the second half of byte one in bit positions 12 to 15 of the LEAC instruction. A "yes" result leads to the address determined in step 128 or 129 to be incremented by the contents of the index register by means of step 131, and a "no" result of step 130 causes control to by-pass step 131.Thus, the address of the second operand of the
LEAC instruction is available, this address is then checked in a number of check steps.

Step 132 checks whether the address of the second operand is a valid address, i.e., whether it is not outside the permissible range of addresses of the data processing system.

If it is not, an address field of type I (invalid) is indicated by step 133. Alternatively, i.e., if the address is a valid address, step 134 determines whether the second operand begins on a full word boundary. Step 135 provides an addressing error indication if step 134 yields a " no " indication. This type of addressing error is classified as a type II error. A "yes" indication leads to a further check step 136, which determines whether the address found is a protected address. If so, step 137 provides a type III addressing error indication, which, in turn, leads to a storage protection interrupt of the current program. Alternatively, step 138 is next im implemented, this checks the operation code of the instruction (bit positions zero to seven) to determine whether it contains "LEAC".

If it does not, step 139 checks the presence of other OP codes by means of operations not shown in detail. These operations are of no interest in this connection, since it is assumed that the instruction to be interpreted is a LEAC instruction. A "yes" indication in step 138 results in the termination of the interpretation cycle and entry into the execution (E) cycle. The first step of this cycle step 140 consists of reading in the second operand. In step 141 the second operand is checked for zero. If the value found is other than zero, the second operand, in step 142, is transferred to register R1 which represents the first operand of the LEAC operation.

Thus, the operation "Load Address" is executed, and the machine can begin to interpret the next instruction. If on the other hand, step 141 yields a " yes", step 143 leads to an interrupt code 0020 to be transferred to a corresponding register (not shown), which, by step 144, causes the current program to be interrupted. The program interrupt of step 144 is identical to the program interrupt of step 155 in Figure 7A and leads to the supervisor program being invoked to execute the operations as previously described with reference to Figure 7B.

The operations as explained by means of
Figure 9 can be executed utilising a suitable microprogram on known microprogrammed
IBM System/360 machines. A system of this type is described, for example, in British
Patent Specification No. 1,242,437.

In the embodiment as described above, the tables EXTAB and ENTAB were formed in a single store containing the respective problem programs. This store would normally be main storage 18. It may, however, be advantageous in the interest of high operat
ing speeds to store the tables in one or several stores operating at particularly high speeds and which respond to instructions referring to ENTRY or EXTRN symbols. The same holds for address table

ADTAB, with the corresponding working storage being controlled by the supervisor program. Data processing systems which, in addition to main storage, comprise high-speed working stores or universally usable register sets are known from the art, as may be seen, for example, from the above mentioned British Patent
Specification.

The arrangements are also suitable for computers which do not use an instruction for externally defined addresses similar to the
LEAC instruction. In such cases the following instruction sequence is to be inserted into the code listings (e.g. listing 1) instead of the
LEAC instruction:
L 10, =A(EX)
LTR 10, 10
BNZ *+6
SVC nnn
These instructions forming part of the standard instructions of IBM System/360 data processing equipment serve the following purpose. By means of the L instruction an address specified more specifically in the second operand is loaded into a register specified by the first operand.The second instruction is a "Load and Test" instruction, by means of which the contents of register 10 are reloaded into this register, with various test operations referring to the numerical1 value to be loaded being executed. One f these test operations checks whether the value to be loaded is zero. Instruction BNZ refers to a branch that is taken if the tested or checked value is other than zero. This leads to the next instruction being skipped if the value tested by means of the LTR instruction is other than zero. Alternatively, instruction SVC is implemented which causes a branch to be taken to the supervisor program. The operations executed by these in structions have the same effect as those effected in response to the LEAC instruction.

WHAT WE CLAIM IS:-
1. A stored program data processing system including an internal store or stores arranged to store program modules which are to be executed under the control of a supervisor program, in which each program module has associated therewith a first table comprising external symbols defined in at least one other module and address fields corresponding to each of these symbols, which fields are initially blank, and a second table comprising internal symbols defined by the module and the addresses of these symbols, and the supervisor program has associated therewith a reference table defining the

**WARNING** end of DESC field may overlap start of CLMS **.